Reasoning about the Executability of Goal-Plan Trees

Yuan Yao[1], Lavindra de Silva[2], and Brian Logan (✉)[3]

[1] School of Computer Science
University of Nottingham
yvy@cs.nott.ac.uk
[2] Institute for Advanced Manufacturing
Faculty of Engineering
University of Nottingham
lavindra.desilva@nottingham.ac.uk
[3] School of Computer Science
University of Nottingham
bsl@cs.nott.ac.uk

Abstract. User supplied domain control knowledge in the form of hierarchically structured agent plans is at the heart of a number of approaches to reasoning about action. This knowledge encodes the "standard operating procedures" of an agent for responding to environmental changes, thereby enabling fast and effective action selection. This paper develops mechanisms for reasoning about a set of hierarchical plans and goals, by deriving "summary information" from the conditions on the execution of the basic actions forming the "leaves" of the hierarchy. We provide definitions of necessary and contingent pre-, in-, and postconditions of goals and plans that are consistent with the conditions of the actions forming a plan. Our definitions extend previous work with an account of both deterministic and non-deterministic actions, and with support for specifying that actions and goals within a (single) plan can execute concurrently. Based on our new definitions, we also specify requirements that are useful in scheduling the execution of steps in a set of goal-plan trees. These requirements essentially define conditions that must be protected by any scheduler that interleaves the execution of steps from different goal-plan trees.

1 Introduction

User supplied domain control knowledge in the form of hierarchically structured agent plans is at the heart of a number of approaches to reasoning about action. This knowledge encodes the "standard operating procedures" of an agent for responding to environmental changes, thereby enabling fast and effective action selection. Various lines of previous work have exploited such control knowledge, including multi-agent coordination [6, 7], interleaved plan execution in single-agent systems [17, 16], heuristic approaches to speeding up classical planning [2,

11, 4], and approaches to synthesising desirable primitive and abstract plans [12, 8].

This paper develops mechanisms for reasoning about a set of hierarchical plans and goals, by deriving "summary information" from the conditions on the execution of the basic actions forming the "leaves" of the hierarchy. We provide definitions of necessary and contingent pre-, in-, and postconditions of goals and plans that are consistent with the conditions of the (possibly nondeterministic) actions forming a plan. Such information is useful when writing agent programs, e.g. when deciding which goal-plan tree is the minimally interfering "building block" to include within a new plan in order to bring about a desired postcondition. In addition to summarising the "static" properties of a single goal-plan tree, we also define requirements that are useful in scheduling the execution of steps in a set of goal-plan trees. While goal-plan trees are most commonly used to represent a BDI agent's domain knowledge, the mechanisms we present could equally be used to represent and reason about the executability of Hierarchical Task Network (HTN) planning [10] structures, e.g., to synthesise new HTN recipes from existing task networks. HTN and BDI systems are closely related in terms of syntax and semantics, making it possible to translate between the two representations [14].

The paper extends the most closely related strands of work in the literature, i.e., [6, 7, 17, 16] in two main ways. Like us, these authors also derive summary information from a set of hierarchical plans, and use that information to find a schedule for the concurrent execution of a given set of top-level goals. Our first extension is an account of both deterministic and non-deterministic primitive actions, and the second is the ability to specify that actions and goals within a (single) plan can execute concurrently. We also contribute novel corresponding definitions for the conditions that must be protected by any scheduler that interleaves the execution of steps from different goal-plan trees.

The remainder of this paper is organised as follows. In Section 2 we discuss closely related work from the literature. In Section 3, we define the 'static', necessary and contingent conditions of actions, plans and goals. Then, in Section 4 we define the corresponding 'dynamic' notions, which specify the conditions that must be taken into account when scheduling. Finally, in Section 5, we conclude and identify directions for future work.

2 Related Work

Our approach is closely related to two previous strands of work in the literature. The first is that of Clement et al. [6, 7], where algorithms are presented for deriving "summary" information from developer-defined hierarchical plans belonging to the agents in a multi-agent system. The derived knowledge is then used to find a schedule that coordinates the activities of the agents at run time. The work of Thangarajah et al. [17, 16, 15] is similar, though they focus on the single-agent case. They describe an approach based on summary information that coordinates the various goal-plan trees of a single agent, in order to exploit positive

interactions between them and to avoid negative interactions, both of which involve reasoning about necessary and possible (summary) conditions of different ways of achieving a goal. They give algorithms for scheduling goal-plan trees, e.g., to determine whether a newly adopted (sub)goal will definitely be safe to execute without conflicts, or will definitely result in conflicts. In the latter case, Thangarajah et al. suspend the goal until it is safe to execute it.

An important difference between the work of Thangarajah et al. and that of Clement et al., is that Thangarajah et al. define the necessary post-condition of a goal or plan as the effects that are necessarily brought about at any (even an intermediate) stage during the goal's or plan's possible executions, whereas Clement et al. define a necessary post-condition as those effects that necessarily hold at the end of all executions. We incorporate both these notions in our approach: our definition of a necessary postcondition of a plan or goal in Section 3 is similar to the necessary postconditions of Clement et al, and the notion of execution conditons that must hold for the successful execution of a set of goal-plan trees we present in Section 4 is similar to the necessary postconditions of Thangarajah et al. Another important difference involves the treatement of cases where a plan step makes the 'descendant' (sub)plan associated with a later plan step inapplicable. Clement et al. assume that such conflicts can be resolved during the scheduling phase, by inserting an available (concurrent) plan—possibly one belonging to a different agent—that asserts a suitable post-condition.[4] We disallow such conflicts, and define a "local" notion of a contingent condition which does not rely on other concurrent plans.

Our work is also related to that of de Silva et al. [9], who focus on how summary information could be used for the synthesis of "abstract plans". In [9], the authors describe how both the strands of work described above could be extended to support agent programs that include variables, i.e., to a restricted first-order language. While the approach of de Silva et al. also supports basic actions, the actions they consider are deterministic and cannot be executed in parallel with other actions. In contrast, our approach is sufficiently general to allow the parallel execution of (nondeterministic) actions and subgoals.

3 Goal-Plan Trees

As in [17, 15] we use *goal-plan trees* to represent the relations between goals, plans and actions, and to reason about the interactions between intentions. The root of a goal-plan tree is a top-level goal[5] (goal node), and its children are the plans that can be used to achieve the goal (plan nodes). Plans may in turn

[4] This assumption is related to the Modal Truth Criterion [5]. See also [23], where scheduling the concurrently executing plans of a single agent is used to recover from action failures.

[5] We assume a procedural interpretation of goals ('goals to do' rather than goals to achieve a state). It is straightforward to adapt the definitions below for declarative goals.

⟨GoalType⟩ ::= ⟨GoalTypeName⟩ ⟨Precondition⟩ ⟨In-condition⟩ ⟨Postcondition⟩ ⟨Plans⟩
⟨GoalTypeName⟩ ::= ⟨Label⟩
⟨Plans⟩ ::= ⟨PlanTypeName⟩ (, ⟨PlanTypeName⟩)*

⟨PlanType⟩ ::= ⟨PlanTypeName⟩ ⟨Precondition⟩ ⟨In-condition⟩ ⟨Postcondition⟩ ⟨PlanBody⟩
⟨PlanTypeName⟩ ::= ⟨Label⟩
⟨PlanBody⟩ ::= ⟨ExecutionStep⟩ (; ⟨ExecutionStep⟩)*
⟨ExecutionStep⟩ ::= ⟨ActionTypeName⟩ | ⟨GoalTypeName⟩ | (⟨ExecutionStep⟩ || ⟨ExecutionStep⟩)

⟨ActionType⟩ ::= ⟨ActionTypeName⟩ ⟨Precondition⟩ ⟨In-condition⟩ ⟨Postcondition⟩
⟨ActionTypeName⟩ ::= ⟨Label⟩

⟨Precondition⟩ ::= ϵ | ⟨Condition⟩ (, ⟨Condition⟩)*
⟨In-condition⟩ ::= ϵ | ⟨Condition⟩ (, ⟨Condition⟩)*
⟨Postcondition⟩ ::= ϵ | ⟨Condition⟩ (, ⟨Condition⟩)*
⟨Condition⟩ ::= ⟨Statement⟩ | *NOT* ⟨Statement⟩
⟨Statement⟩ ::= *string* | ⟨Variable⟩ = ⟨Value⟩
⟨Label⟩ ::= *unique string*
⟨Variable⟩ ::= *unique string*
⟨Value⟩ ::= *string*

⟨GoalInstance⟩ ::= ⟨InstanceName⟩ ⟨GoalType⟩
⟨PlanInstance⟩ ::= ⟨InstanceName⟩ ⟨PlanType⟩
⟨ActionInstance⟩ ::= ⟨InstanceName⟩ ⟨ActionType⟩
⟨InstanceName⟩ ::= ⟨Label⟩

Fig. 1. BNF Syntax of goal-plan trees with actions

contain subgoals (goal nodes), giving rise to a tree structure representing all possible ways an agent can achieve the top-level goal.

In [17, 15] goal-plan trees contain only goals and plans. We extend their definition of goal-plan trees to allow primitive actions in plans in addition to subgoals as in [23, 21]. Plans thus consist of a sequence of steps, where a step is either a primitive action or a subgoal, or a parallel composition of plan steps.[6] Parallel composition is supported by BDI agent systems such as JACK [20] and HTN-like planning systems such as RETSINA [13].

Figure 1 shows the BNF syntax of our extended goal-plan trees. A *GoalType* is a template for a goal. A *GoalInstance* is created when an agent chooses to pursue a particular instance of goal-type. Similarly, a *PlanType* is a template for a plan, and a *PlanInstance* is created when the agent executes a particular plan. In addition, we introduce an *ActionType* as a template for an action, and an *ActionInstance* is created when a particular action is chosen for execution by the agent. *GoalTypeName*, *PlanTypeName* and *ActionTypeName* are labels that

[6] The goal-plan trees in [23, 21] do not include parallel constructs.

indicate the type of the goal, the plan or the action respectively. *Plans* represents the set of plan-types that may be used to satisfy a goal of the corresponding *GoalType*.

Goals, plans and actions have pre-, in-, and postconditions. Pre- and postconditions specify respectively the states of the environment which must hold immediately before the action, plan, or goal is executed, and which are brought about by executing the action, plan, or goal. In-conditions specify the states of the environment which must hold for the duration of the execution of the action, plan, or goal. In-conditions of plans and goals are thus relevant when their associated actions are interleaved or overlapped, and in-conditions of actions are relevant when they are overlapped.

We model the environment using a set of propositions Φ, and define pre-, in- and postconditions of a goal-plan tree node η (an action, plan or goal) as sets of literals (elements of $\Phi^+ = \Phi \cup \{\neg p \mid p \in \Phi\}$) as follows.

Precondition: a precondition is a set of literals $\phi = pre(\eta)$, $\phi \subseteq \Phi^+$ that must be true for η to begin execution (where η is an action or plan), or for η to be achieved (where η is a goal).

In-condition: an in-condition is a set of literals $\upsilon = in(\eta)$, $\upsilon \subseteq \Phi^+$ that must hold during the execution of η (where η is an action or plan), or during the pursuit of η (where η is a goal); if any of the literals in υ becomes false during execution, the action, plan or goal is aborted with failure.

Postcondition: a postcondition (or effect) is a set of literals $\psi = post(\eta)$, $\psi \subseteq \Phi^+$ that are or may be made true by executing η (where η is an action or plan), or by achieving η (where η is a goal).

We distinguish two types of pre-, in- and postconditions: necessary and contingent. A *necessary* (or universal) condition must hold for all executions of an action or plan or for all ways of achieving a goal, while a *contingent* (or existential) condition must hold for some executions of the action or plan or some ways of achieving the goal. We denote the necessary and contingent preconditions as $pre_n(\eta)$ and $pre_c(\eta)$, where η is an action, plan or goal, and stipulate that $pre(\eta) = pre_n(\eta) \cup pre_c(\eta)$. Similarly, we denote necessary and contingent in-conditions as $in_n(\eta)$ and $in_c(\eta)$, and necessary and contingent postconditions as $post_n(\eta)$ and $post_c(\eta)$, and stipulate that $in(\eta) = in_n(\eta) \cup in_c(\eta)$ and $post(\eta) = post_n(\eta) \cup post_c(\eta)$. The necessary and contingent postconditions of an action, plan, or goal are always disjoint, and the same applies to necessary and contingent in-conditions, and to necessary and contingent preconditions.

While the relevant pre-, in- and postconditions form part of the definition of an action, the conditions of plans and goals are derived from the conditions of their actions and subgoals (in the case of plans) and from plans to achieve the goal (in the case of goals). We give formal definitions of necessary and contingent conditions for actions, plans, and goals in the sections below.

3.1 Actions

Actions are the basic steps an agent can perform in order to change its environment. Actions may be deterministic or non-deterministic. Deterministic actions

have a single outcome (postcondition), while the execution of a non-deterministic action results in one of a set of possible outcomes (set of postconditions).

The precondition of an action α, $pre_n(\alpha) = \phi$, is always necessary (and $pre_c(\alpha) = \emptyset$). The in-condition of an action, $in_n(\alpha) = \upsilon$, is also necessary (and $in_c(\alpha) = \emptyset$). Deterministic actions have a single postcondition ψ. The necessary postcondition of a deterministic action α is defined as $post_n(\alpha) = \psi$, and the contingent postcondition is defined by $post_c(\alpha) = \emptyset$. The execution of a non-deterministic action results in one of a set of possible postconditions $\{\psi_1, \ldots, \psi_n\}, \psi_i \subseteq \Phi^+$. The necessary postcondition of a non-deterministic action α is defined as $post_n(\alpha) = \bigcap \psi_i \in \{\psi_1, \ldots, \psi_n\}$, and the contingent postcondition is defined by $post_c(\alpha) = \bigcup \psi_i \in \{\psi_1, \ldots, \psi_n\} \setminus post_n(\alpha)$.[7] We assume that action specifications are consistent in the sense that each possible outcome of an action is itself consistent, i.e., that $\psi_i \not\models \bot, 1 \leq i \leq n$, and that execution of an action does not invalidate the in-condition of the action, i.e., $in_n(\alpha) \cup post_n(\alpha) \cup post_c(\alpha) \not\models \bot$.[8]

3.2 Plans

A plan π consists of a sequence of actions, subgoals, and parallel compositions of actions and subgoals. That is, a plan is of the form $\pi = \alpha_1; \ldots; \alpha_m$, where each α_i is either an action, a subgoal or a parallel composition $\beta_1 \| \ldots \| \beta_k$, where each β_i is either an action or a subgoal. In the interests of generality, we make no assumptions about the execution of a parallel composition of actions and subgoals: steps $\beta_1, \ldots, \beta_k$ may be executed in parallel, i.e., they may overlap in any of the ways defined in [1], or their execution may be arbitrarily interleaved. For example, if β_i is an action and β_j a subgoal, then β_i may be interleaved with the actions appearing in the goal-plan tree for β_j. However, we require that there are no conflicts between the pre-, in- and postconditions of $\beta_1, \ldots, \beta_k$ and, as a result, the overall postcondition of the parallel composition is "stable", i.e., for each $\beta_i, \beta_j, 1 \leq i \leq k, 1 \leq j \leq k, i \neq j$, the necessary and contingent postconditions of β_i must be consistent with the necessary and contingent pre- in- and postconditions β_j.

More precisely, the necessary postcondition of a parallel composition $\alpha = \beta_1 \| \ldots \| \beta_k$ is defined as the union of the necessary postconditions of each of its parallel steps (which, as above, are assumed to be "non-conflicting"):

$$post_n(\alpha) = \bigcup_{i=1}^{k} post_n(\beta_i).$$

The contingent postcondition of a parallel composition is defined similarly, except that we exclude any contingent postcondition literal of a step if it is also a

[7] Note that this means the necessary conditions of an action may differ from its contingent conditions.

[8] For entailment, we sometimes treat a a set of literals as the conjunction of the literals in the set.

necessary postcondition of some other step, i.e.,

$$post_c(\alpha) = \bigcup_{i=1}^{k} post_c(\beta_i) \setminus post_n(\alpha).$$

However, the definition of the necessary pre- and in-conditions of a parallel composition must to take into account the necessary and contingent postconditions of steps that may establish—by virtue of how steps may be interleaved or overlapped—the pre- and in-conditions of other steps, i.e.,

$$con_n(\alpha) = \bigcup_{i \in \{1,\ldots,k\}} \big(con_n(\beta_i) \setminus \bigcup_{j \in \{1,\ldots,k\} \setminus \{i\}} (post_n(\beta_j) \cup post_c(\beta_j)) \big),$$

where *con* is either *pre* or *in*. That is, the necessary in-conditions of steps that may be established by interleaving of other steps in the parallel composition are not considered necessary. Finally, the contingent pre- (resp. in-) conditions of a parallel composition are the contingent pre- (resp. in-) conditions of the parallel steps, together with the necessary pre- (resp. in-) conditions of its steps that may be established by other steps. That is, necessary in-conditions that may be established by interleaving of other steps in the parallel composition become contingent. Formally, we define

$$con_c(\alpha) = \bigcup_{i=1}^{k} \big(con_c(\beta_i) \cup con_n(\beta_i) \big) \setminus con_n(\alpha)$$

where *con* is either *pre* or *in*.

We can now define the necessary and contingent pre-, in- and postconditions of plans. The necessary precondition of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as

$$pre_n(\pi) = pre_n(\alpha_1) \cup \bigcup_{i=2}^{m} \left[pre_n(\alpha_i) \setminus \bigcup_{j=1}^{i-1} post_n(\alpha_j) \cup post_c(\alpha_j) \right]$$

that is, the necessary preconditions of steps that are not established by the necessary or contingent postconditions of previous steps. Necessary preconditions must hold for all executions of π.[9] Note that we do not assume that a plan establishes all the preconditions of the steps in the plan. For example, a plan to make coffee may assume that the agent is in the kitchen and that there is coffee in the kitchen. However, we do assume that each plan π ensures a 'free-choice' among its 'descendant' plans (plans that achieve the subgoals of π). For example, a plan to make coffee should not cause the agent to leave

[9] As we are concerned with the executability of plans rather than their applicability in a particular context, we do not include the context condition (belief context) of a plan specified by a developer to be part of its precondition. However, in a well-formed plan, the necessary precondition should form (part of) the context condition of the plan.

the kitchen before the coffee is made, as that would then invalidate one or more subplans, e.g. one that grinds the coffee. More precisely, for any step α_k in a plan $\pi = \alpha_1; \ldots; \alpha_n$, if there is an earlier step α_i $(i < k)$ and a literal $l \in post_n(\alpha_i)$ such that $\sim l \in pre_n(\alpha_k) \cup pre_c(\alpha_k) \cup in_n(\alpha_k) \cup in_c(\alpha_k)$, then there is also an intermediate step α_j $(i < j < k)$ with $\sim l \in post_n(\alpha_j) \cup post_c(\alpha_j)$, where $\sim l = \neg p$ if $l = p$ and $\sim l = p$ if $l = \neg p$.

If π contains non-deterministic actions or subgoals, it may also have contingent preconditions, i.e., preconditions which may have to be established depending on the outcome of a non-deterministic action (if the outcome of the action fails to achieve the precondition of a later action in the plan) or the choice of plan to achieve a subgoal. Thus, the contingent precondition of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as $pre_c(\pi) = pre_c(\alpha_1) \cup$

$$\bigcup_{i=2}^{m} \left[\Big(pre_c(\alpha_i) \setminus \bigcup_{j=1}^{i-1} post_n(\alpha_j) \Big) \cup \Big(\big(pre_n(\alpha_i) \setminus \bigcup_{j=1}^{i-1} post_n(\alpha_j) \big) \cap \bigcup_{j=1}^{i-1} post_c(\alpha_j) \Big) \right].$$

That is, the possible preconditions of each step not established by necessary postconditions of previous steps, and the necessary preconditions of each step that are (possibly) established by contingent postconditions of previous steps, but not by their necessary postconditions. Observe that sets $pre_n(\pi)$ and $pre_c(\pi)$ are mutually exclusive by definition.

The necessary in-condition of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as

$$in_n(\pi) = \bigcup_{i=1}^{m-1} (in_n(\alpha_i) \cap in_n(\alpha_{i+1})).$$

That is, a necessary in-condition of a plan π is an in-condition that is necessary for two or more consecutive steps in π. The rationale for this definition arises from the role of in-conditions in scheduling. The in-condition of an action α specifies which other actions may be scheduled in a parallel with the action without negative interactions—only actions α' whose postcondition does not result in a negative interaction with the in-condition of α may be scheduled in parallel with α. When reasoning with summary information at the plan level, we seek to detect situations where the parallel or interleaved execution of actions in a plan π' may result in a negative interaction between the postcondition of an action in π' and the in-condition of actions in π'.

The contingent in-condition of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as

$$in_c(\pi) = \bigcup_{i=1}^{m} \big(in_c(\alpha_i) \cup in_n(\alpha_i) \big) \setminus in_n(\pi).$$

Finally, we define the necessary and contingent postconditions of a plan. The necessary postconditions of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as

$$post_n(\pi) = \{l \mid \exists i : l \in post_n(\alpha_i) \wedge \forall j \in \{i, \ldots, m\} :\sim l \notin post_n(\alpha_j) \cup post_c(\alpha_j)\}.$$

That is, the necessary postconditions of each step not 'undone' by the necessary or contingent postconditions of later steps. The contingent postcondition of a plan $\pi = \alpha_1; \ldots; \alpha_m$ is defined as $post_c(\pi) = post_c^1(\pi) \cup post_c^2(\pi)$, where

$$post_c^1(\pi) = \{l \mid \exists i : l \in post_c(\alpha_i) \wedge \forall j \in \{i, \ldots, m\} :\sim l \notin post_n(\alpha_j)\}$$

and

$$\begin{aligned} post_c^2(\pi) = \{l \mid \exists i : l \in post_n(\alpha_i) \wedge \\ \exists j \in \{i, \ldots, m\} :\sim l \in post_c(\alpha_j) \wedge \\ \forall j \in \{i, \ldots, m\} :\sim l \notin post_n(\alpha_j)\}. \end{aligned}$$

That is, the contingent postcondition of a plan is either a contingent postcondition of a step that is not 'undone' by the necessary postcondition of a later step, or a necessary postcondition of a step that may be 'undone' by a contingent postcondition of a later step. Observe that sets $post_n(\pi)$ and $post_c(\pi)$ are mutually exclusive by definition.

3.3 Goals

A goal γ is associated with a set of plans $\pi_1, \ldots, \pi_n$ that achieve γ, and the pre-, in- and postconditions of γ are derived from this set of associated plans. For simplicity, we stipulate that goals with the same *GoalType* as γ do not appear in the goal-plan tree rooted at γ.[10]

The necessary pre-, in- and postconditions of a goal γ associated with plans $\pi_1, \ldots, \pi_n$ is defined as

$$con_n(\gamma) = \bigcap_{i=1}^{n} con_n(\pi_i),$$

where *con* is either *pre*, *in* or *post*. That is, necessary pre-, in-, or postconditions must hold respectively before, during, or after all ways of achieving γ.

The contingent pre-, in-, and postconditions of a goal γ associated with plans $\pi_1, \ldots, \pi_n$ is defined as

$$con_c(\gamma) = \bigcup_{i=1}^{n} con_c(\pi_i) \cup \left[\bigcup_{j=1}^{n} con_n(\pi_j) \setminus con_n(\gamma) \right]$$

where *con* is either *pre*, *in* or *post*. That is, a pre-, in-, or postcondition is contingent for γ if it is a contingent condition of a plan π_i to achieve γ, or if it is a necessary condition for a plan π_j but not for γ itself (i.e., it is a necessary condition of some but not all plans for γ.)

The definitions above capture the relationship between the pre-, in- and postconditions of actions, plans and goals in a goal-plan tree. The conditions for actions define which propositions must be true before, during and after either all

[10] This is a standard assumption in computing summary information e.g., [6, 7, 17, 16]. The assumption can be relaxed, but the definitions of conditions below become more complex.

executions of an action (necessary conditions), or some execution of the action (contingent conditions). The conditions for plans define which propositions must be true before, during and after either all executions of a plan, or some execution of the plan. The necessary preconditions of a plan specify the states in which the plan is applicable. The conditions for goals define which propositions must be true before, during and after either all means of achieving a goal or some means of achieving a goal.

4 Execution Conditions

In the previous section, we defined the necessary and contingent conditions for the execution of a single goal-plan tree. In this section, we consider information relevant to the execution of a *set* of goal plan trees.

If an agent always executes at most one goal-plan tree at a time, e.g., it executes its intentions in first-in-first-out order, then the execution conditions are the same as those given in Section 3. However, in many application domains, an agent's goal-plan trees comprising a system or an agent's user supplied domain knowledge are executed in parallel. For example, in many BDI agent architectures, the plans comprising the agent's intentions are executed in parallel, e.g., by executing one step of an intention at each cycle in a round robin fashion [3, 20]. Interactions between interleaved steps in plans in different goal-plan trees may result in conflicts, i.e., the execution of a step in one plan makes the execution of a step in another concurrently executing plan impossible.

Given a set of goal-plan trees, the *scheduling problem* is to determine which step of which goal-plan tree to execute next, so as to minimise the number of execution conflicts.[11] Scheduling aims to minimise the number of plan failures resulting from choices made by the agent regarding the order of execution of a set of goal-plan trees, thus allowing the largest number of goals to be achieved.[12] Our aim here is not to solve the scheduling problem; for example, we do not consider the problem of which plan an agent should adopt for a given (sub)goal—this is the concern of deliberation scheduling. Rather, we focus on defining conditions that must or may hold on all possible future executions of a set of goal-plan trees. As such, the conditions we define should be taken into account by any scheduler, but are neutral with respect to the actual form of deliberation scheduling adopted. It turns out that, in our setting, the information relevant for scheduling differs from the conditions on the well-formedness of a goal-plan tree defined

[11] Scheduling may also be used to maximise the number of positive interactions between goal-plan trees, as in, e.g., [17, 23]; we do not consider positive interactions here.

[12] Plans may fail for reasons that are outside the control of the agent, e.g., due to changes in the environment, or actions of other agents violating the conditions of a plan. Several approaches, e.g., [18, 19, 21] have been proposed which attempt to avoid such failures. However, the information about goal-plan trees required by these approaches (essentially the the percentage of world states for which there is some applicable plan for any subgoal within an intention) is different from that required for scheduling, and we do not consider them further here.

in the previous section. The definitions of execution conditions below therefore depart from those in, e.g., [16, 15].

To define the execution conditions for a goal-plan tree, we need some auxiliary notions. Given a set of goal-plan trees $T = \{\tau_1, \ldots, \tau_n\}$, an *execution context* for T is a set of pairs $I = \{(\tau_1, \rho_1), \ldots, (\tau_n, \rho_n)\}$, where each ρ_i defines the set of possible future execution paths for τ_i. Each ρ_i corresponds to the point execution has reached in the goal-plan tree τ_i, and hence the possible paths future execution of τ_i may follow. (I essentially corresponds to the intentions of a BDI agent.) Initially, each ρ_i points to the top-level goal of the corresponding goal-plan tree τ_i. As execution of τ_i proceeds, plans are selected, restricting the possible future execution paths to a subtree of τ_i captured by ρ_i. In the interests of brevity, and where no confusion can arise, we shall refer to possible future execution paths simply as possible execution paths.

An initial set of possible execution paths ρ_0 for a goal-plan tree τ is a sequence $(\pi_i, \alpha_1), \ldots, (\pi_i, \alpha_k)$, where $\pi_i = \alpha_1; \ldots; \alpha_k$ is the selected plan for the top-level goal of τ. As execution progresses, a set of possible execution paths $\rho = (\pi_1, \alpha_1), (\pi_2, \alpha_2), \ldots, (\pi_m, \alpha_m)$ evolves as follows. The successor set of possible execution paths ρ' of ρ is $(\pi_2, \alpha_2), \ldots, (\pi_m, \alpha_m)$ if α_1 is an action, and $\rho' = (\pi'_1, \alpha'_1), \ldots, (\pi'_1, \alpha'_n), (\pi_2, \alpha_2), \ldots, (\pi_m, \alpha_m)$ if α_1 is a subgoal γ_1 and $\pi'_1 = \alpha'_1; \ldots; \alpha'_n$ is the plan selected for γ_1. Only sets of possible execution paths which are the initial set of possible execution paths in τ (corresponding to the top-level of goal of τ) or are obtained by the progression step described above are sets of possible execution paths in τ.

We can now define the necessary and contingent execution conditions of an execution context. Informally, the necessary execution conditions of a set of possible execution paths ρ_i, are those conditions that must hold or be achieved at some point in all possible future executions of a goal-plan tree τ_i starting from ρ_i, and the contingent execution conditions are those conditions that must hold or be achieved at some point of time in at least one possible future execution (but not all executions) of τ_i starting from ρ_i. When executing the set of goal-plan trees in T in parallel, such execution conditions must be protected — if the execution conditions of two sets of possible execution paths ρ_i and ρ_j intersect, then interleaving steps in ρ_i and ρ_j may result in conflicts.

4.1 Actions

As actions are atomic, the necessary and contingent execution conditions of an action α are identical to the corresponding necessary and contingent conditions for α (we denote execution conditions with a $*$):

$$con^*_n(\alpha) = con_n(\alpha) \qquad con^*_c(\alpha) = con_c(\alpha)$$

where con^*_n and con_n are either pre^*_n and pre_n, in^*_n and in_n or $post^*_n$ and $post_n$ respectively, and similarly con^*_c and con_c are either pre^*_c and pre_c, in^*_c and in_c or $post^*_c$ and $post_c$.

4.2 Plans

The necessary and contingent execution conditions of a plan π differ from the corresponding necessary and contingent conditions for π. As steps in plans in different goal-plan trees may be arbitrarily interleaved, we need to protect *all* the preconditions in a plan, even if they are established by a preceding step in the same plan, as the condition may be invalidated by a step in a plan in another goal-plan tree.

The necessary execution pre-, in- and postcondition of a parallel composition $\alpha = \beta_1 \| \ldots \| \beta_k$ is therefore the union of the necessary execution conditions of each β_i, i.e.,

$$con_n^*(\alpha) = \bigcup_{i=1}^{k} con_n^*(\beta_i)$$

where con_n^* is either pre_n^*, in_n^* or $post_n^*$.

The contingent pre-, in- and post- execution conditions of a parallel composition is also defined as the union of contingent execution conditions of each β_i, except that we exclude any contingent postcondition literal of a step if it is also a necessary postcondition of some other step, i.e.,

$$con_c^*(\alpha) = \bigcup_{i=1}^{k} con_c^*(\beta_i) \setminus con_n^*(\alpha).$$

The necessary and contingent execution preconditions of a plan (or plan suffix) $\pi = \alpha_1; \ldots; \alpha_m$ are therefore given by

$$pre_n^*(\pi) = \bigcup_{i=1}^{m} pre_n^*(\alpha_i) \qquad pre_c^*(\pi) = \bigcup_{i=1}^{m} pre_c^*(\alpha_i) \setminus pre_n^*(\pi).$$

Similarly, the postconditions of interest are no longer the 'eventual' postconditions of the plan, since the postcondition of an action α_i 'undone' by a later step $\alpha_j, i < j$ in π may be 'visible' to a step in a plan in another goal-plan tree. The necessary and contingent execution postconditions of π are therefore given by

$$post_n^*(\pi) = \bigcup_{i=1}^{m} post_n^*(\alpha_i) \qquad post_c^*(\pi) = \bigcup_{i=1}^{m} post_c^*(\alpha_i) \setminus post_n^*(\pi).$$

In contrast, the necessary and contingent execution in-conditions of π are the same as the necessary and contingent in-conditions of π: $in_n^*(\pi) = in_n(\pi)$, $in_c^*(\pi) = in_c(\pi)$. (Since $in_n(\pi)$ and $in_c(\pi)$ define conditions that must hold between the execution of steps in π, they also apply to the interleaving of plan steps.)

4.3 Goals

As with plans, the necessary and contingent execution conditions of a goal γ associated with plans $\pi_1, \ldots, \pi_n$ differ from the corresponding necessary and

contingent conditions for γ. (The conditions of goals are defined in terms of the conditions of their associated plans.)

The necessary pre-, in- and post- execution conditions of a goal γ associated with plans $\pi_1, \ldots, \pi_n$ is defined as

$$con_n^*(\gamma) = \bigcap_{i=1}^{n} con_n^*(\pi_i),$$

where *con* is either *pre*, *in* or *post*. That is, necessary pre-, in-, or post- execution conditions must hold respectively before, during, or after all ways of achieving γ.

The contingent pre-, in-, and post- execution conditions of a goal γ associated with plans $\pi_1, \ldots, \pi_n$ is defined as

$$con_c^*(\gamma) = \bigcup_{i=1}^{n} con_c^*(\pi_i) \cup \left[\bigcup_{j=1}^{n} con_n^*(\pi_j) \setminus con_n^*(\gamma) \right],$$

where *con* is either *pre*, *in* or *post*. That is, a pre-, in-, or postcondition is contingent for γ if it is a contingent condition of a plan π_i to achieve γ, or if it is a necessary condition for a plan π_j but not for γ itself (i.e., it is a necessary condition of some but not all plans for γ.)

4.4 Sets of Execution Paths

We can now define the necessary and contingent execution conditions of a set of possible execution paths $\rho = (\pi_1, \alpha_1), \ldots, (\pi_k, \alpha_k)$ of a goal-plan tree τ. These conditions can be used to reason about possible conflicts that may arise in the execution of each pair of goal-plan trees in a set of goal-plan trees.

The necessary execution precondition of a set of possible execution paths ρ is given by

$$pre_n^*(\rho) = \bigcup_{i=1}^{k} pre_n^*(\alpha_i).$$

That is, we must protect the necessary preconditions of all steps in ρ. The contingent execution precondition of ρ is given by

$$pre_c^*(\rho) = \bigcup_{i=1}^{k} pre_c^*(\alpha_i) \setminus pre_n^*(\rho).$$

Contingent preconditions are those that may need to be established during execution, depending on the choice of plan to achieve a goal.

The necessary execution in-condition of a set of possible execution paths ρ is given by

$$in_n^*(\rho) = \bigcup_{i=1}^{k} in_n^*(\alpha_i) \cup \bigcup_{i=1}^{k} in_n(\pi_i).$$

That is, we must protect the in-conditions of all steps in ρ, and in addition we also need to protect the in-conditions of all currently executing plans in ρ. The contingent execution in-condition of ρ is given by

$$in_c^*(\rho) = \bigcup_{i=1}^{k} in_c^*(\alpha_i) \cup \bigcup_{i=1}^{k} in_c(\pi_i) \setminus in_n^*(\rho).$$

The necessary and contingent execution postconditions of a set of possible execution paths ρ is given by

$$post_n^*(\rho) = \bigcup_{i=1}^{k} post_n^*(\alpha_i) \qquad post_c^*(\rho) = \bigcup_{i=1}^{k} post_c^*(\alpha_i) \setminus post_n^*(\rho).$$

Finally, the necessary execution conditions of a set of possible execution paths ρ are given by

$$cond_n^*(\rho) = pre_n^*(\rho) \cup in_n^*(\rho) \cup post_n^*(\rho),$$

and the contingent execution conditions of ρ are given by

$$cond_c^*(\rho) = pre_c^*(\rho) \cup in_c^*(\rho) \cup post_c^*(\rho).$$

Conflicts may occur when we have complementary literals in the execution conditions of two sets of possible execution paths, ρ_i and ρ_j, i.e., when

$$\exists\, l \in cond_x^*(\rho_i) \wedge \sim l \in cond_x^*(\rho_j),$$

where $cond_x^*$ is either $cond_n^*$ or $cond_c^*$. Clearly, there are different cases. For example, conflicts between the necessary execution conditions of two execution paths may be a more serious problem than conflicts between contingent execution conditions.

If no conflicts (as defined above) occur between two sets of possible execution paths ρ_i and ρ_j, then the next step in either (or both) ρ_i and ρ_j may be safely executed. On the other hand, if there are conflicts between the two sets of possible execution paths, then we could still interleave their execution such that they do not interfere with one another, e.g., by borrowing techniques from [16]. For example, if the conflict between ρ_i and ρ_j is due to complementary literals in $in_n^*(\rho_i)$ and $post_n^*(\rho_j)$, then we could delay the execution of ρ_j until ρ_i progresses to a point where there is no longer a conflict with ρ_j. This is because ρ_j might otherwise interfere with the in-condition of a plan that is currently being pursued.[13] If the conflict between ρ_i and ρ_j is due to complementary literals in

[13] Note that if ρ_i and ρ_j are considered in order of the priority of the associated top-level goal (or ties are broken arbitrarily), deadlock (as defined in [16, 15]) cannot arise, even if there are complementary literals in $in_n^*(\rho_j)$ and $post_n^*(\rho_i)$. However, this may result in conditions of the lower priority set of possible execution paths being violated. In such cases, more sophisticated intention scheduling techniques, e.g., [22, 21] may be able to find an interleaving that protects the conditions of both sets of possible execution paths.

$pre_c^*(\rho_i)$ and $post_c^*(\rho_j)$, an optimistic approach would be to first execute ρ_j until it progresses to a point where a conflict no longer occurs with ρ_i, and only then begin executing ρ_i. This assumes that either execution of ρ_j does not actually bring about the conflicting literal, or that if it is brought about, execution of ρ_i is such that the conflicting contingent precondition is not required, or a step within ρ_i itself asserts the negation of the conflicting literal.

5 Conclusion and Future Work

This paper has provided definitions of pre-, in-, and postconditions of actions, plans, and goals, for an extended goal-plan tree that supports the execution of steps (goals and actions) in parallel, as well as the specification of both deterministic and non-deterministic actions. Our definitions essentially capture 'static' and 'dynamic' notions of conditions, which are derived from the primitive ones specified within the basic actions that form plans. We believe that 'static' properties defined by our notions will facilitate authoring agent programs, particularly because it is important to know the properties of the individual "building blocks" (goal-plan trees) that are available when composing a new plan. We have used our 'dynamic' notions of conditions to derive the conditions that must be protected by any scheduler, when interleaving two or more goal-plan trees.

We foresee two main directions for future work. First, we could allow for a step in a plan to necessarily invalidate one or more (though not all) 'descendant' (sub)plans of a later step, and accordingly extend our notions of the necessary and contingent postconditions of a plan. This extension would involve identifying which postconditions in the later step are never asserted due to the conflict (and are thereby neither necessary nor contingent postconditions), and which ones are always asserted due to the conflict, by virtue of certain descendant plans being always inapplicable. Second, we could explore how to generate a schedule for interleaving two or more goal-plan trees, while respecting the conditions that we have identified as needing to be protected.

References

1. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
2. Jorge A. Baier, Christian Fritz, and Sheila A. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 26–33, 2007.
3. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.
4. Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

5. David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.
6. Bradley J. Clement and Edmund H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 495–502, 1999.
7. Bradley J. Clement, Edmund H. Durfee, and Anthony C. Barrett. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research (JAIR)*, 28:453–515, 2007.
8. Lavindra de Silva, Sebastian Sardina, and Lin Padgham. First Principles Planning in BDI systems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1105–1112, 2009.
9. Lavindra de Silva, Sebastian Sardina, and Lin Padgham. Summary information for reasoning about hierarchical plans. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 1300–1308, 2016.
10. Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1123–1128, 1994.
11. Christian Fritz, Jorge A. Baier, and Sheila A. McIlraith. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for planning and beyond. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 600–610, 2008.
12. Subbarao Kambhampati, Amol Dattatraya Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 882–888, 1998.
13. Massimo Paolucci, Onn Shehory, Katia P. Sycara, Dirk Kalp, and Anandeep Pannu. A planning component for RETSINA agents. In *International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL),*, pages 147–161. Springer-Verlag, 2000.
14. Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1001–1008, 2006.
15. John Thangarajah and Lin Padgham. Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, 47(1):17–56, 2011.
16. John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 721–726, 2003.
17. John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting and exploiting positive goal interaction in intelligent agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 401–408, 2003.
18. John Thangarajah, Sebastian Sardina, and Lin Padgham. Measuring plan coverage and overlap for agent reasoning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1049–1056, 2012.
19. Max Waters, Lin Padgham, and Sebastian Sardina. Evaluating coverage based intention selection. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 957–964, 2014.
20. Michael Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.

21. Yuan Yao and Brian Logan. Action-level intention selection for BDI agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, pages 1227–1235, Singapore, May 2016. IFAAMAS, IFAAMAS.
22. Yuan Yao, Brian Logan, and John Thangarajah. SP-MCTS-based intention scheduling for BDI agents. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 1133–1134, 2014.
23. Yuan Yao, Brian Logan, and John Thangarajah. Robust execution of BDI agent programs by exploiting synergies between intentions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2558–2564, Phoenix, USA, February 2016. AAAI, AAAI Press.